Using COBOL at the NIH Computer Center

September 1998

National Institutes of Health
Center for Information Technology
NIH Computer Center
12 South Drive MSC 5607
Bethesda, Maryland 20892-5607

Publication No. CIT003

Table of Contents

1 INTRODUCTION

This manual describes the use of the COBOL/MVS programming language at the NIH Computer Center. This manual is intended to give programmers the COBOL information they need in order to create new programs and to maintain programs running on the MVS South System. The information in this manual should be used in conjunction with the *NIH Computer Center User's Guide*, *Batch Processing and Utilities at NIH*, and the manuals described in Section 3.

The COBOL/MVS programming language receives full (Level 1) support. Questions on COBOL should be directed to the Technical Assistance and Support Center (TASC), either by phone at (301) 594-3278 or by submitting a Problem Tracking Report (PTR). There are several methods of submitting a PTR:

- World Wide Web
  Users with NIHnet or Internet connections can submit a PTR through the World Wide Web. To access the PTR system, connect to:

  http://datacenter.cit.nih.gov/ptr.html

- Electronic Mail
  PTRs can also be submitted to the Computer Center by sending electronic mail to the WYLBUR initials PTR or the Internet address PTR@CU.NIH.GOV. Mailed PTRs must have a valid SUBJECT header containing the submitter's name and telephone number, and be of the form

  Subject: PTR FROM name TELEPHONE phone-number

  For example:

  Subject: PTR FROM Tom Jones TELEPHONE 6-1111

- ENTER PTR
  Users can submit a PTR through WYLBUR's ENTER PTR command.

Changes that affect the use of the COBOL/MVS language will be fully tested and pre-announced through the *Interface* newsletter. For a full description of Level 1 support, see the *NIH Computer Center User's Guide*.

COBOL (COmmon Business Oriented Language) is a programming language, similar to English, which is designed for data processing oriented programming applications. The Computer Center uses the IBM COBOL/MVS compiler and libraries. This compiler is compatible with the American National Standard for COBOL. It supports the 1985 standards

and the 1989 addendum to the 1985 standards; 1985 standards is the default. Language Environment for OS/390 & VM (LE) provides the run-time environment for COBOL/MVS.

There is a Federal Information Processing Standard (FIPS) for this language. Certain special features and extensions of the language fall outside the FIPS standard. Programs written using only FIPS approved features can be transported more readily between federal installations and different vendors' mainframes. Federal policy encourages the use of features within FIPS standards. COBOL/MVS statements not conforming to the 1985 FIPS COBOL standards are identified in the manual *IBM COBOL for MVS & VM Compiler and Run-Time Migration Guide*, GC26-4764.

For information on Year 2000 COBOL Conversion Services, go to:
http://silk.nih.gov/silk/year2000/

1.1 Procedure Names

Note: the Binder now performs the link-editing functions previously performed by the Linkage Editor.

The procedure names used in this manual are:

CBL3COMP
CBL3OBJ
CBL3LKGO
CBL3LDGO
CBL3LKMM
CBL3LKSM
CBL3CALL

Each procedure name follows the pattern:

lllvffff

where	"lll"	is the language prefix (CBL for COBOL)
	"v"	is the version (3 for COBOL)
	"ffff"	is the function

The meaning of each function is given below:

COMP	compilation only
OBJ	compile and store object module

LKGO	use the Binder (formerly the Linkage Editor) and execute program
LDGO	use the Loader and execute program
LKMM	use the Binder to store a link-edited load module into an existing multi-member PDS
LKSM	use the Binder to store a link-edited load module into a new single-member PDS
CALL	execute a fully link-edited load module

In the examples throughout this manual, the following conventions apply:

“aaaa”	the account number
“iii”	the programmer’s registered initials
“dsname”	name of data set
“progname”	name of program stored in partitioned data set (PDS)
“fileser”	volume serial number of disk where data set is located; required only if the data set is not cataloged
“primary”	primary quantity requested in the SPACE parameter
“blocks”	number of directory blocks
“stepname”	name of step which executes the procedure; should be unique within a job
“ddname”	user-supplied ddname; should be unique within job step

2 DCB INFORMATION FOR SYSOUT DATA SETS

Listed in the figure below are the default record formats and blocksizes for all SYSOUT data sets in the COBOL/MVS procedures.

PROCEDURE NAME	STEP NAME	DD NAME	DEFAULT RECFM/BLKSIZE
CBL3COMP	COMP	SYSPRINT	FBA 121
CBL3LKGO	LOAD GO	SYSPRINT SYSOUT PUNCH	FA 121 FBA 120 F 80
CBL3OBJ	COMP	SYSPRINT	FBA 121
CBL3LKSM	LOAD	SYSPRINT	FA 121
CBL3LKMM	LOAD	SYSPRINT	FA 121
CBL3CALL	GO	SYSOUT PUNCH	FBA 120 F 80
CBL3LDGO	GO	SYSLOUT SYSOUT PUNCH SYSDBOUT	FBSA 121 FBA 120 F 80 FA 121

Figure 1. SYSOUT DCB Information for COBOL/MVS Procedures

3 COBOL PUBLICATIONS

The CIT Technical Information Office distributes general information, technical and vendor publications and certain software to the user community. *Using COBOL at the NIH Computer Center* is one of the many publications available online through the World Wide Web at:

http://datacenter.cit.nih.gov/cfb.pub.txt.html

Users may order publications in the following ways:

- Using the World Wide Web, visit:

  http://livewire.nih.gov/publications/publications.asp

  and select the option for ordering publications online. Some publications may not be available through this ordering system.

- Sign on to WYLBUR and use the ENTER PUBWARE command to order publications.
- If you cannot order a publication online, you may place an order by visiting TASC in Building 12A or by telephone.

The following manuals relevant to COBOL/370 can be ordered:

COBOL for MVS & VM Compiler & Run-Time Migration Guide, GC26-4764
: This publication provides information to help users move their run time to IBM Language Environment for MVS & VM (Language Environment) and to upgrade their source programs to COBOL for OS/390 & VM or COBOL for MVS & VM.

Language Environment for OS/390 & VM Debugging Guide & Run-Time Messages, SC28-1942
: This IBM document provides assistance with detecting and locating programming errors that occur during run time under Language Environment. It can help establish a debugging process to analyze data and narrow the scope and location of where an error might have occurred.

IBM COBOL Report Writer Precompiler Programmer Manual, SC26-4301
: This publication provides information about the Report Writer statements. It is intended for programmers engaged in the writing of new programs using Report Writer or the maintenance of old ones.

IBM COBOL for MVS & VM Language Reference, SC26-4769
: This manual describes the language elements of the COBOL language.

IBM COBOL for MVS & VM Programming Guide, SC26-4767

This manual describes the techniques and nuances of programming with the COBOL Language. It explains how to compile, link-edit and execute or compile and load a COBOL program with COBOL/MVS.

Interface

This is a series of technical notes for users, published by the Computer Center. All changes to Computer Center standards and facilities are announced in this publication.

4 COBOL/MVS COMPILER OPTIONS

The standard options generated for the COBOL/MVS compiler are listed below.

APOST	Indicates to the compiler that the apostrophe (‘) is acceptable as the character to delineate literals, and to use that character in the generation of figurative constants.
BUFSIZE(4096)	Specifies the amount of the SIZE parameter to be reserved for compiler data set buffers.
DATA(31)	(31) Above/below 16-megabyte line or from unrestricted storage.
FLAG(I)	Print all warning diagnostics as well as error diagnostics.
LANGUAGE(EN)	Indicates the output will be printed in mixed case English.
LINECOUNT(60)	Number of lines printed per page of source listing.
NOADV	The first byte must be reserved for the carriage control character for files with WRITE ADVANCING.
NOAWO	APPLY WRITE-ONLY clause will not be in effect.
NOCMPR2	Generate code that conforms to COBOL 85 standards.
NOCOMPILE(S)	Causes normal compilation with both syntax checking and object code generation.
NOCURRENCY	Specifies that no alternate default currency symbol will be used; the default currency symbol is the dollar sign ($).
NODBCS	Will not cause the compiler to use X’OE’ and X’OF’ (SI) as shift codes.
NODECK	No object deck will be punched.
NODUMP	The compiler should produce an informative message rather than a dump if it encounters a D-level (“disaster”) error condition during its processing.
NODYNAM	Indicates that subprograms will be link-edited with the calling program into a single load module.
NOEVENTS	Does not produce/update event files.

NOEXIT	Will not allow the compiler to accept user-supplied modules in the place of SYSIN, SYSLIB (or copy library), and SYSPRINT.
NOFASTSRT	Only conforms to the COBOL 85 Standard and does not allow DFSORT to perform input and output.
NOFLAGMIG	Non-ANSI 85 standard statements will not be flagged.
NOFLAGSAA	No COBOL/MVS System Application Architecture flagging.
NOFLAGSTD	Level or subset of COBOL will not be specified.
NOLIB	Indicates that no COPY or BASIS request will be part of the COBOL/MVS source input stream. If library facilities are to be used, LIB must be specified at compilation time.
NOLIST	Assembler Language listing of procedure division is not to be generated.
NOMAP	Glossary and global tables are not to be printed.
NONAME	Indicates that programs compiled in a batch environment will be link-edited into a single load module.
NONUMBER	Indicates that line numbers have not been recorded in the source text and that the compiler should generate source numbers for use with error messages as well as in LIST, OFFSET, XREF.
NOOFFSET	Condensed listing of PROCEDURE DIVISION is not to be listed.
NOOPTIMIZE	Indicates that optimized object code is not to be generated.
NORENT	Generates a non-reentrant object module.
NOSSRANGE	Will not generate code to check subscript.
NOTERM	Indicates that progress and diagnostic messages are not to be printed on the SYSTERM terminal data set.
NOTEST	Indicates that the program cannot be debugged at the terminal using the Interactive Debug Facility.

NOVBREF	A brief summary/cross reference of verbs used in the source program and a count of how often each verb appeared will not be provided.
NOWORD	Does specify alternate reserved word table.
NOXREF	A sorted cross-reference of data and procedure names is not to be listed.
NUMPROC(PFD)	Bypasses invalid sign processing.
OBJECT	Places generated object code on disk or tape to be later used as input for the Binder.
OUTDD(SYSOUT)	Indicates that SYSOUT is the ddname of the file to be used for debug output and for data when SYSOUT is specified, either implicitly or explicitly, in a DISPLAY statement.
SEQUENCE	Compiler will check card sequence of source module.
SIZE(1024000)	Specifies the amount of main storage available to the compiler.
SOURCE	Source module is to be listed.
SPACE(1)	Single space compiler listing.
TRUNC(OPT)	Applies to the movement of COMPUTATIONAL arithmetic fields. When NOTRUNC is specified, movement of items is dependent on the size of the field (halfword, fullword).
ZWB	Instructs the compiler to strip the sign from a signed external decimal field when comparing this field to an alphanumeric field. If ZWB is specified, the signed external decimal field is moved to an intermediate field, in which its sign is removed before it is compared to the alphanumeric field. NOZWB should be used when, for example, input numeric fields are to be compared with spaces.

For further information on the above options and possible other options, refer to the *IBM COBOL for MVS & VM Programming Guide*, SC26-4767.

If any options are to be changed, the user must specify them in the EXEC statement. The OPTIONS symbolic parameter may be used in place of the PARM parameter. The use of the OPTIONS symbolic parameter is illustrated in the examples in this manual.

For those who must override or augment the cataloged procedures, the stepnames used in the procedure are given in each section.

5 COMPILING AND RUNNING COBOL/MVS PROGRAMS

The procedures in this section are used to compile, link-edit (using the Binder), and execute COBOL/MVS programs.

5.1 Using the Compiler

The COMP procedure provides the user with a one-step procedure to compile COBOL/MVS source code for diagnostic messages, and, if compilation is successful, to prepare the input for further processing (e.g., the LKGO procedure). This procedure stores the output of the compiler into a temporary data set to be used later in the job and then deleted.

Symbolic Parameters for CBL3COMP

Required	**Value to be supplied**
None	None
Optional	**Value to be supplied**
OPTIONS=parms	Compiler parameters

The internal stepname for the CBL3COMP procedure is COMP.

Example 1:

To compile only.

```
//stepname  EXEC  CBL3COMP
//COMP.SYSIN  DD *
   (source program)
```

Example 2:

To compile and obtain a procedure division map (Assembler language listing) and a data division map.

```
//stepname  EXEC  CBL3COMP,OPTIONS='LIST,MAP'
//COMP.SYSIN  DD  *
   (source program)
```

5.2 Using the Binder

> Note: the Binder now performs the link-editing functions previously performed by the Linkage Editor.

The LKGO procedure performs the following functions:

- link-edits the program to prepare a load module for execution
- executes the load module.

The LKGO procedure provides the user with the DD statements needed to use the printer (SYSOUT), the card punch (PUNCH), the sort/merge messages data set (SORTMSGS), verb execution count data set (SYSCOUNT), and the COBOL debug data set (SYSDBOUT). Users must provide their own JCL for any additional I/O units (data sets) used. Section 6.4 discusses specifying user-defined libraries with LKGO.

There must be one GO.ddname DD statement describing each data set used. DD statements to override ddnames within the procedure must precede those for ddnames to be added to the procedure. If more than one DD statement is being overridden, the override statements must be in the same order as the existing DD statements in the procedure. See *Batch Processing and Utilities at NIH* for a description of the format of DD statements.

Symbolic Parameters for CBL3LKGO

Required	**Value to be supplied**
None	None

Optional	**Value to be supplied**
OPTIONS=parms	Binder parameters
CORE=nnnnK	Region for GO step; 4096K is the default
LIBNAME='aaaaiii.dsname'	Dsname of first user-defined library
LIBDISK=fileser	Volume for first library; required only if the data set is not cataloged
LIBSTOR=type	Unit name for first library; FILE is the default
ALTNAME='aaaaiii.dsname'	Dsname of second user-defined library
ALTDISK=fileser	Volume for second library; required only if the data set is not cataloged
ALTSTOR=type	Unit name for second library; FILE is the default

The stepnames within the CBL3LKGO cataloged procedure are LOAD for the link-edit step and GO for the run step.

When a COBOL/MVS main program and its subroutines are compiled in the same job, the main program should be compiled first unless there is a specific entry point coded in the main COBOL/MVS program.

Example 3:

To compile the main program and execute it.

```
//stepname  EXEC  CBL3COMP
//COMP.SYSIN DD *
   (source program)
//stepname  EXEC  CBL3LKGO
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

Example 4:

To compile the main program and execute it. The OPTIONS symbolic parameter in the compile step requests a procedure division map and a data division map. The OPTIONS parameter in the run step requests the Binder option XREF. The CORE parameter supplies a larger region size for the GO step.

```
//stepname  EXEC  CBL3COMP,OPTIONS='LIST,MAP'
//COMP.SYSIN DD *
   (source program)
//stepname  EXEC  CBL3LKGO,CORE=nnnnK,OPTIONS=XREF
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
```

When executing the program in a separate step such as in the CBL3LKGO procedure, only the program-defined execution parameters and LE-defined execution options may be passed. In this case all LE-defined options are specified after the last slash. Program-defined parameters are placed before the slash.

Example 5:

To pass the LE-defined option DEBUG using the LKGO procedure.

```
//stepname EXEC CBL3LKGO,
//  PARM.GO='/DEBUG'
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * (if needed)
   (data)
   (data)
```

5.3 Creating and Using Object Modules

The OBJ procedure is used to compile source code and store the resultant object module into a sequential data set. The output of this procedure must be processed by the Binder before it can be run. The LKGO procedure may be used to link-edit and execute the object module(s) created by an OBJ procedure.

Symbolic parameters for CBL3OBJ

Required	**Value to be supplied**
NAME='aaaaiii.dsname'	Dsname of object module to be stored
Optional	**Value to be supplied**
DISK=fileser	Required only for a data set written to a dedicated disk
STORAGE=type	Unit name for the object module; FILE is the default
OPTIONS=parms	Compiler parameters
STATUS=status	NEW is the default; use OLD to replace an existing data set
SIZE=primary	Primary space allocation for object module; default is 500 (enough for approximately 500 source statements)
UNITS=type	Allocation units for object module; the default is blocks of 1024 bytes

The internal stepname for the CBL3OBJ procedure is COMP.

Example 6:

To compile and save the object module.

```
//stepname EXEC CBL3OBJ,NAME='aaaaiii.dsname'
//COMP.SYSIN  DD  *
    (source program)
```

Example 7:

To compile and save into an existing data set. Former contents will be destroyed.

```
//stepname EXEC CBL3OBJ,STATUS=OLD,
//   NAME='aaaaiii.dsname'
//COMP.SYSIN DD *
   (source program)
```

Example 8:

To compile and save overriding the default for primary space allocation. If there are more than 500 source statements, the 'primary' value should be roughly equal to the number of statements in the program.

```
//stepname EXEC CBL3OBJ,SIZE=primary,
//   NAME='aaaaiii.dsname'
//COMP.SYSIN DD *
   (source program)
```

To execute a program which has been stored by an OBJ procedure, use the CBL3LKGO procedure. The user must supply a //LOAD.SYSLIN DD statement describing the data set containing the program which was compiled and saved.

Example 9:

To compile, save the object module, link-edit, and run. The object module saved as "aaaaiii.dsname1" from the OBJ procedure is used as input for the link-edit step.

```
//stepname EXEC CBL3OBJ,NAME='aaaaiii.dsname1'
//COMP.SYSIN DD *
   (source program)
//stepname EXEC CBL3LKGO
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,DISP=SHR
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * ( if needed)
   (data)
```

Example 10:

To execute a main program and subroutines which have been created as separate data sets by the OBJ procedure. The user must supply a DD statement for each data set that contains a program or subroutine and insure that the main program is defined first.

```
//stepname EXEC CBL3LKGO
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,
//   DISP=SHR
//  DD DSN=aaaaiii.dsname2,DISP=SHR
//  DD DSN=aaaaiii.dsname3,DISP=SHR
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

Example 11:

To execute a program where the main program is to be compiled and the subroutines have been stored by the OBJ procedure in two data sets. These data sets will be concatenated with the data set created by the COMP step.

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC CBL3LKGO
//LOAD.SYSLIN DD
// DD DSN=aaaaiii.dsname1,DISP=SHR
// DD DSN=aaaaiii.dsname2,DISP=SHR
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

5.4 Using the Loader

The LDGO procedure combines the link-edit and run steps into one. The Loader will accept object modules and load modules. It will also search libraries defined by the SYSLIB DD statement within the procedure if unresolved external references remain after processing the primary input defined by the SYSLIN DD statement within the procedure. DD statements are provided for use of the printer (SYSPRINT and SYSOUT), the card punch (PUNCH), the verb execution count data set (SYSCOUNT), and the sort/merge messages data set (SORTMSGS).

The LDGO procedure should be used during the early stages of program development (debugging); it is particularly recommended for the development of small and medium-sized programs. Using LDGO is often more economical than using LKGO, but a dump from a

LDGO run may not be sufficient to resolve a problem. If so, the job may have to be rerun using the Binder (LKGO).

Additional technical information on the use of the Loader is given in *Batch Processing and Utilities at NIH*.

Symbolic Parameters for CBL3LDGO

Required	**Value to be supplied**
None	None
Optional	**Value to be supplied**
OPTIONS=parms	Loader and GO parameters
CORE=nnnK	Region for GO step; 300K is the default
EPT=entry	Entry point for main program; the default is the value in PROGRAM-ID
IBNAME='aaaaiii.dsname'	Dsname of first user-defined macro library
LIBDISK=fileser	Volume for first library; required only if the data set is not cataloged
LIBSTOR=type	Unit name for first library; FILE is the default
ALTNAME='aaaaiii.dsname'	Dsname of second user-defined macro library
ALTDISK=fileser	Volume for second library; required only if the data set is not cataloged
ALTSTOR=type	Unit name for second library; FILE is the default

The procedure name is CBL3LDGO; the internal stepname is GO.

Example 12:

To compile the main program and use the Loader to execute it.

```
//stepname  EXEC  CBL3COMP
//COMP.SYSIN  DD  *
   (source program)
//stepname  EXEC  CBL3LDGO
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * (if needed)
   (data)
```

When using the Loader for a program that requires execution time options, slashes are important. The Loader procedure consists of only one step, therefore, there is only one execute statement to pass all necessary Loader options, program-defined execution parameters, and COBOL-defined execution options. All Loader options are specified before the first slash; all LE-defined options are specified after the last slash.

Example 13:

To use the Loader to pass 'WEDNESDAY' as the value of the program-defined parameter. The execution time option DEBUG is limited to the tracking of 40 procedure names.

```
//stepname  EXEC  CBL3LDGO,
//  OPTIONS='/WEDNESDAY/DEBUG=(40)'
//GO.ddname  DD etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

Example 14:

To turn the Loader MAP option off and pass 'WEDNESDAY' as the value of the program-defined parameter.

```
//stepname EXEC CBL3LDGO,
//  OPTIONS='NOMAP/WEDNESDAY/'
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * (if needed)
   (data)
```

6 STORING AND USING PROGRAMS IN USER LIBRARIES

> Note: the Binder now performs the link-editing functions previously performed by the Linkage Editor.

The following procedures were developed to store user programs in load module form. Users can develop and maintain their own private libraries which are partitioned data sets.

A Partitioned Data Set (PDS) is composed of one or more sequential “members”, each of which may be accessed independently. Each member has a unique name, up to 8 characters long, stored in a directory. The directory contains an entry for each member consisting of the member name and a pointer to the location of the member in the data set. When a member is deleted or replaced, only the member-name-pointer is deleted or changed. The space used by the member cannot be reused until the data set is condensed. If there is not enough space for a new or replacement member, or if there are no more free entries in the directory, no members can be added. A job that attempts to add a new member to a PDS which is full usually ABENDs with a X37 completion code. A PDS must be stored on a disk and cannot exceed one disk pack in size.

Load modules (the output from the Binder) must be stored in PDSs. The programs may be either fully or partially link-edited. The Binder will automatically search libraries defined by the SYSLIB DD statement to resolve calls or references to programs that are not included in the main input stream defined by the SYSLIN DD card. The libraries are searched in the order they are defined. When a reference is found, no further searching is done, and the next search begins again at the first library. If all external references and subroutine calls are resolved, the program is fully link-edited and is, therefore, directly executable without link-editing again. If the external references and calls are not to be resolved, the NCAL option must be specified in the EXEC statement for the procedure used to store the program. The program is then partially link-edited and must be reprocessed by the Binder before it can be executed.

Executing fully resolved load modules may cost less because a link-edit step is saved every time the program is run; however, problems may develop as a result of updates to the computer system. Fully resolved load modules cannot take advantage of some of these system improvements. In addition, a program may fail to run if it contains old interfaces to system modules.

To avoid these problems, fully resolved load modules should be re-created periodically, particularly whenever a new system release is installed. If recreating the fully resolved modules is difficult, it may be better to keep partially resolved modules and do the final link-edit each time the program is run.

6.1 Storing Programs in Single-Member User Libraries

The LKSM procedure is used to link-edit and store a load module (output of the Binder) into a single-member partitioned data set (PDS). The COMP and OBJ procedures may be used to prepare input for the LKSM procedure. A short step, executed before the link-edit step, deletes the PDS if it already exists. Then the link-edit step creates the new data set. If the data set does not already exist, the delete step issues a message, but does not affect later processing. If the output library is to be created on the MSS, STORAGE=MSS must be specified. Additionally, the symbolic parameters SIZE, UNITS and INCR must be coded with the appropriate values for requesting space on the MSS.

The user may define two private call libraries for resolving external references. They are searched in their order of concatenation; if members with duplicate names exist, the first one found will be selected. The private libraries are searched after the COBOL/MVS language library and before NIH.UTILITY.

Symbolic Parameters for CBL3LKSM

Required	**Value to be supplied**
NAME='aaaaiii.dsname'	Dsname of PDS to receive load module
Optional	**Value to be supplied**
DISK=fileser	Volume for PDS; required only if the data set is not cataloged
STORAGE=type	Unit name for PDS; FILE is the default
OPTIONS=parms	Binder parameters
PROGRAM=progname	Member name for load module; the default is MAIN
SIZE=primary	Primary space allocation for load module; the default is 100 units
UNITS=type	Allocation units for load module; the default is blocks of 1024 bytes
INCR=secondary	Number of units in each secondary allocation; the default is 12
STEPEND=disp	Disposition for the load module; the default is KEEP
UNUSED=	Nullifying causes retention of unused space; the default is RLSE
INDEX=blocks	Number of directory blocks for load module PDS; the default is 1
LIBNAME='aaaaiii.dsname'	Dsname of first user-defined library
LIBDISK=fileser	Volume for first library; required only if the data set is not cataloged
LIBSTOR=type	Unit name for first library; FILE is the default
ALTNAME='aaaaiii.dsname'	Dsname of second user-defined library
ALTDISK=fileser	Volume for second library; required only if the data set is not cataloged

ALTSTOR=type	Unit name for second library; FILE is the default

The internal stepnames for the CBL3LKSM procedure are SCRATCH, for the step to scratch the data set if it already exists, and LOAD, for the link-edit step.

Example 15:

To compile, fully link-edit, and store a program into a single-member PDS.

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
    (source program)
//stepname EXEC CBL3LKSM,NAME='aaaaiii.dsname'
```

Example 16:

To compile, fully link-edit, and store a program, overriding the default space allocation. If the program requires more than the default space allocation, the SIZE parameter should be used. The default SIZE parameter allows the user to obtain at least 10 tracks for the load module (unneeded space is released).

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
    (source program)
//stepname EXEC CBL3LKSM,NAME='aaaaiii.dsname',
//  SIZE=primary
```

Example 17:

To fully link-edit, and store a main program and subroutines using input from the OBJ procedure. The user must supply a DD statement for each data set that contains a program or subroutine and insure that the main program is defined first.

```
//stepname EXEC CBL3LKSM,NAME='aaaaiii.dsname'
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,DISP=SHR
// DD DSN=aaaaiii.dsname2,DISP=SHR
// DD DSN=aaaaiii.dsname3,DISP=SHR
```

Example 18:

To compile a main program (CBL3COMP), link-edit using subroutines previously compiled with the OBJ procedure, and create a fully resolved single-member load module (CBL3LKSM).

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC CBL3LKSM,NAME='aaaaiii.dsname'
//LOAD.SYSLIN DD
//  DD  DSN=aaaaiii.dsname1,DISP=SHR
//  DD  DSN=aaaaiii.dsname2,DISP=SHR
```

6.2 Storing Programs In Multi-Member User Libraries

The procedures described below enable the user to add programs to multi-member partitioned data sets and execute them. Before using these procedures, refer to the manual *Batch Processing and Utilities at NIH* for information on how to establish and maintain partitioned data sets. These procedures differ from the OBJ and LKSM procedures in that many programs can be stored in one data set. The OBJ and LKSM procedures store only one program in a data set.

The LKMM procedure adds a program to a private partitioned data set. If the program name already exists in the data set, it will be replaced. The Binder input is the same as for the LKGO procedure.

The user may define two private call libraries for resolving external references. They are searched in their order of concatenation; if members with duplicate names exist, the first one found will be selected. The private libraries are searched after the COBOL/MVS language library and before NIH.UTILITY. If no libraries are to be searched (no external references are to be resolved), OPTIONS=NCAL must be specified for the LKMM step; this creates a partially link-edited load module.

Symbolic Parameters for CBL3LKMM

Required	**Value to be supplied**
NAME='aaaaiii.dsname'	Dsname of PDS to receive load module
PROGRAM=progname	Program name; member name in PDS
Optional	**Value to be supplied**
DISK=fileser	Volume for PDS; required only if the data set is not cataloged
STORAGE=type	Unit name for PDS; FILE is the default
OPTIONS=parms	Binder parameters

LIBNAME='aaaaiii.dsname'	Dsname of first user-defined library
LIBDISK=fileser	Volume for first library; required only if the data set is not cataloged
LIBSTOR=type	Unit name for first library; FILE is the default
ALTNAME='aaaaiii.dsname'	Dsname of second user-defined library
ALTDISK=fileser	Volume for second library; required only if the data set is not cataloged
ALTSTOR=type	Unit name for second library; FILE is the default

The stepname within the CBL3LKMM cataloged procedure is LOAD.

Example 19:

To create a multi-member PDS on a FILE volume and then compile and add a partially link-edited program to the PDS. The program must be fully link-edited along with all of its subroutines, as shown in the next example, before it is executed.

```
//  EXEC  PGM=IEFBR14
//NEWPDS  DD  DSN=aaaaiii.dsname,DISP=(NEW,CATLG),
//            UNIT=FILE,SPACE=(TRK,(10,2,3))
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC CBL3LKMM,NAME='aaaaiii.dsname',
// PROGRAM=progname,OPTIONS=NCAL
```

Example 20:

To fully link-edit and add a program to a cataloged PDS where one or more of the routines is being compiled. The same PDS is used to resolve external references; therefore, LIBNAME and NAME refer to the same data set.

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
  (source program)
//stepname EXEC CBL3LKMM,LIBNAME='aaaaiii.dsname',
// NAME='aaaaiii.dsname',PROGRAM=progname
//LOAD.SYSLIN DD
//  DD  *
    INCLUDE  SYSLIB(main program name)
    ENTRY entryname
```

The INCLUDE and ENTRY cards are control statements to the Binder. They always begin after column 1. The INCLUDE statement is used to define as input to the Binder modules that would not automatically be brought in. The ENTRY statement indicates the starting point of the program.

These control statements and the two preceding DD statements are not needed in this example if the main program is one of the routines being compiled. In general, the ENTRY statement is not needed for COBOL/MVS if the main program is the first input to the Binder or if it is in object module form.

Example 21:

To fully link-edit and add a program to a PDS, where the main program and its subroutines were previously stored in the same PDS as partially link-edited load modules. If 'progname' and 'main progname' are the same, the partially link-edited main program will be replaced.

```
//stepname CBL3LKMM,NAME='aaaaiii.dsname',
//  PROGRAM=progname,
//  LIBNAME='aaaaiii.dsname'
//LOAD.SYSLIN DD *
    INCLUDE SYSLIB(main progname)
```

Example 22:

To link-edit and execute a program where the main program and some subroutines are in two separate PDSs and other subroutines are being compiled.

```
//stepname  EXEC  CBL3COMP
//COMP.SYSIN  DD  *
   (source program)
/*
//stepname EXEC CBL3LKGO,
//  LIBNAME='aaaaiii.dsname1',
//  ALTNAME='aaaaiii.dsname2'
//LOAD.SYSLIN DD
//  DD  *
  INCLUDE SYSLIB(main program name)
  ENTRY entryname
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * (if needed)
   (data)
```

6.3 Using Programs from User Libraries

The CALL procedure is used to execute a fully link-edited program. This procedure provides the user with the DD statements needed to use the printer (SYSOUT), the card punch (PUNCH), the verb execution count data set (SYSCOUNT), the sort/merge messages data set (SORTMSGS), and the COBOL debug data set (SYSDBOUT). These DD statements are the same ones supplied in the CBL3LKGO procedure. The user must supply any additional DD statements required for proper execution of the program.

Symbolic Parameters for CBL3CALL

Required	**Value to be supplied**
NAME='aaaaiii.dsname'	Dsname of PDS containing load module
Optional	**Value to be supplied**
DISK=fileser	Volume for PDS; required only if the data set is not cataloged
STORAGE=type	Unit name for PDS; FILE is the default
PROGRAM=progname	Member name for load module; the default is MAIN
CORE=nnnnK	Region for GO step; 4096K is the default

The stepname within the CBL3CALL cataloged procedure is GO.

Example 23:

To execute a program which has been previously stored by an LKSM procedure.

```
//stepname EXEC CBL3CALL,NAME='aaaaiii.dsname'
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * (if needed)
   (data)
```

Example 24:

To execute a fully link-edited program stored in a PDS on the MSS.

```
//stepname EXEC CBL3CALL,NAME='aaaaiii.dsname',
//  PROGRAM=progname
//GO.ddname  DD  (as many as needed)
//GO.SYSIN  DD  * (if needed)
  (data) cards
```

When executing the program in a separate step such as in the CBL3CALL procedure, only the program-defined execution parameters and LE-defined execution options may be passed. In this case all LE-defined options are specified after the last slash.

Example 25:

To pass 'WEDNESDAY' as the value of the program defined parameter using the CALL procedure.

```
//stepname EXEC CBL3CALL,
//  NAME='aaaaiii.dsname',PARM.GO='WEDNESDAY/'
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  * (if needed)
   (data)
```

6.4 Link-editing from a User Library (Using the Binder)

User-defined libraries can be specified to be searched in resolving external references. Both the Binder and the Loader offer this facility. The symbolic parameter LIBNAME defines the first such library. ALTNAME is available if it is necessary to define a second private library. These private libraries are searched in their order of concatenation; if members with duplicate names exist, the first one found will be selected. The private libraries are searched after the COBOL/MVS language library and before NIH.UTILITY.

Symbolic Parameters for CBL3LKGO and CBL3LDGO

Required	**Value to be supplied**
None	None

Optional	**Value to be supplied**
OPTIONS=parms	Binder or Loader parameters
CORE=nnnnk	Region for GO step; defaults are 4096K for LKGO and 300K for LDGO
EPT=entry	Entry point for main program (Loader only); defaults to PROGRAM-ID value
LIBNAME='aaaaiii.dsname'	Dsname of first user-defined library
LIBDISK=fileser	Volume for first library; required only if the data set is not cataloged
LIBSTOR=type	Unit name for first library; FILE is the default
ALTNAME='aaaaiii.dsname'	Dsname of second user-defined library
ALTDISK=fileser	Volume for second library; required only if the data set is not cataloged
ALTSTOR=type	Unit name for second library; FILE is the default

Example 26:

To use the Binder when a main COBOL/MVS program is compiled and its subroutines are stored as load modules in a private user library.

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC CBL3LKGO,LIBNAME='aaaaiii.dsname'
//GO.ddname DD etc.
```

Example 27:

To use the Loader when a main COBOL/MVS program is compiled and some subroutines are stored as load modules in a private user's libraries.

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC CBL3LDGO,
//    LIBNAME='aaaaiii.dsname1',
//    ALTNAME='aaaaiii.dsname2'
//GO.ddname DD etc.
```

The LKGO procedure may also be used to relink-edit and execute a partially resolved load module stored in a partitioned data set.

The examples above assume the subroutines were stored using the same names they are called by. If these names are not the same, INCLUDE statements must be supplied for the subroutines.

Example 28:

To link-edit and execute a main program and its subroutines which have been partially link-edited and stored into a PDS.

```
//stepname EXEC  CBL3LKGO,
//  LIBNAME='aaaaiii.dsname'
//LOAD.SYSLIN DD *
  INCLUDE  SYSLIB(main program name)
/*
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

Example 29:

To link-edit and execute a program where the main program and some subroutines are in two separate PDSs and other subroutines are being compiled.

```
//stepname EXEC CBL3COMP
//COMP.SYSIN DD *
    (source program)
/*
//stepname EXEC CBL3LKGO,
//  LIBNAME='aaaaiii.dsname1',
//  ALTNAME='aaaaiii.dsname2'
//LOAD.SYSLIN DD
//  DD  *
  INCLUDE SYSLIB(main program name)
  ENTRY entryname
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN DD  *  (if needed)
   (data)
```

7 COBOL/MVS PROGRAMMING AND RUNNING TIPS

In addition to the pointers given below, also consult the "Programming Techniques" section of the *IBM COBOL for MVS & VM Programming Guide*, SC26-4767.

7.1 Restrictions

- The following features of COBOL/MVS cannot be used at this facility:
    - CHECKPOINT/RESTART (the RERUN clause is tolerated)
    - User label handling functions are not supported. Therefore, the label handling format of USE is invalid. The data-mname option of the LABEL RECORDS clause is invalid. USER-DECLARATIVE is never invoked for labeling.
    - Any COBOL/MVS instruction that causes a message to be written on the operator's console (e.g. DISPLAY ON CONSOLE or STOP literal).
- COBOL/MVSM allows the SEGMENTATION feature; you will not improve storage allocation by using it, because COBOL/MVS does not perform overlay.
- When using the SORT verb, always use DISP=SHR in the JCL rather than DISP=OLD for the data set SYS1.SORTLIB. Also, the following statement must be included for the SORT/MERGE program messages if Computer Center procedures are not being used:

```
//stepname.SORTMSGS  DD  SYSOUT=A
```

- COBOL/MVS Interactive DEBUG is not available.

7.2 Program Design and Efficiency

The hints in this section apply to all types of COBOL programs run at NIH.

7.2.1 File Manipulation

- Arrange the Data Division so that the most active files are first. This is particularly important if there are more than five files being used.
- Group FDs by function and define files that are used together in sequence.
- Since core is initialized to zeroes, uninitialized variables may not cause a program to ABEND and may instead cause a program to yield incorrect results. The fact that core is initialized to zeroes should not be depended on by the programmer to initialize data items within the program. System software often modifies core for its own purpose that is later used in the program's region.
- Whenever possible, use one OPEN statement when opening multiple files. This will cause their buffers to be close together and improve locality of reference. Whenever possible, use one OPEN or CLOSE statement for all files.

- Avoid the use of SAME RECORD AREA since this causes buffers to be separated from files not using the option.
- Group references to DIRECT files because buffers for these files are separate from sequential files.
- It is a good practice to CLOSE a file after the last reference to it instead of waiting until job-end time. This will eliminate extra paging in a case where a DCB has been paged out and must be paged in when the data set is closed.
- A FILE STATUS key of 92 represents a logic error in the program processing the file: Typical logic errors include:
    - attempting to read an unopened file
    - attempting to read beyond end-of-file
    - attempting to close a closed file
    - attempting to open an opened file.

  To obtain more information about the FILE STATUS key, see the *IBM COBOL for MVS & VM Programming Guide,* SC26-4767.

7.2.2 Data Definition

- Group items in WORKING-STORAGE according to their usage; items used together should be defined together.
- Define large tables either at the beginning or end of WORKING-STORAGE. Defining a large table in the middle results in unnecessarily separating other items by a large span of addresses.
- Use VALUE clauses for initialization of WORKING-STORAGE items whenever possible. This will provide for initialization at OBJECT time and eliminates execution time statements and potential unnecessary paging.
- Arrange the WORKING-STORAGE Section so that the most commonly used items are in the first 4096 bytes.
- Optimization and better locality of reference may be achieved by carefully selecting data types used. For example, avoid data conversion (and use of conversion subroutines) by comparing like items: COMP to COMP; COMP-3 to COMP-3.
- The generated code for conversion from external decimal (PIC 9) to binary (COMP) is particularly lengthy. Convert on a one-time basis if possible, e.g., move decimal data to a binary field and use it multiple times. External decimal data is always converted either to COMP or COMP-3 before it is used in an arithmetic operation.
- Always give COMP-3 items an uneven PICTURE length. An even PICTURE will cause an extra instruction to be generated.
- Define items used as counters as COMP, to avoid use of conversion routines.

- Use one-byte alpha-numeric data items for all switches and flags (PICTURE X).
- Avoid the use of figurative constants (SPACE OR ZERO) to initialize a multi-character data field:

  77 G PIC X(3).
  MOVE ‘bbb’ to G.

  instead of:
  MOVE SPACE to G.

- All arithmetic items should be signed.
- If the value of a subscript is being frequently incremented without looking up a table entry, use subscripts. If each time the subscript is incremented a value is looked up in the table, use indexing.

7.2.3 Procedure Division

- Avoid numeric comparison of items of unequal number of decimal places.
- Avoid use of ON SIZE ERROR; instead, allow enough decimal places to contain the maximum field. Also avoid use of ROUNDED if possible.
- Use MULTIPLY instead of DIVIDE when possible:

  MULTIPLY A BY .5 GIVING Q
  instead of:
  DIVIDE A BY 2 GIVING Q

- Use ADD instead of MULTIPLY.

  ADD A TO A
  instead of:
  MULTIPLY 2 BY A

- Make all operands have the same number of decimal places for ADD and SUBTRACT statements:

  77 H PIC S9(3)V99
  ADD 3.50 TO H
  instead of:
  ADD 3.5 TO H

- Never multiply by ten or any multiple of ten; instead use a REDEFINES statement.

  77 F PIC S9(5)V99.
  77 F10 REDEFINES F PIC S9(6)V9.

- When using IF statements put the most likely condition first in an IF/OR and the least likely first in an IF/AND.
- Avoid the use of NEXT SENTENCE after the ELSE option of IF statements.
- Try not to use negative compound IF statements.
- Avoid the use of PERFORM statements where the PERFORMed paragraph is not close to the PERFORM statement. In-line code is better if not too large or too frequent.
- It is better to pass a single item to a subroutine (an 01 level with several elementary items) than to pass many individual items.
- Use the absolute value for subscripting whenever possible. If you want the first occurrence of TAX-REC use TAX-REC (1) instead of MOVE 1 to SUB, TAX-REC (SUB).
- Define numeric items used as subscripts as binary.
- Avoid the use of multi-level subscripted or indexed tables if at all possible.

7.2.3.1 Carriage Control

- The WRITE BEFORE ADVANCING and WRITE AFTER ADVANCING options cause a carriage control character to be written in the first position of each record as follows:

AFTER ADVANCING	ASA
BEFORE ADVANCING	machine

  If both BEFORE ADVANCING and AFTER ADVANCING are specified, machine control characters are written.

7.3 Hints

- ABENDs in Language Environment OS/390 & VM: An ABEND is still a problem of some type and the application will be notified of such a problem. There have been some changes in the way in which COBOL/MVS programs run. They now run in what is called a common execution environment, the actual name is Language Environment OS/390 & VM or LE for short. LE sets up the environment which includes acquiring memory, handling errors, and other functions. Before OS/VS COBOL did all these functions itself, and if it ran into a problem it simply gave the user a dump. With COBOL/MVS and LE you actually get more explicit information. COBOL/MVS identifies the problem even down to the statement level and notifies LE it found an application error. LE does not ABEND like OS/VS COBOL did, however it passes back to the user a return code indicating the application did not run as designed. In most cases a return code of U4038 will be issued. The U4038 means the application ended with a software raised or user raised condition of severity 2 or greater. The Computer Center has set the LE parameters to ensure a dump will be created when this happens.

- S001 ABEND - You might not see the S001 return code, however the return code will be something other than zero, probably U4038. Whether you get the original S001 error or U4038 really depends on who traps the error. If COBOL identifies the error you will get an IGZ message and a U4038 return code. If LE or maybe IMS traps the error you will probably continue to get the S001. Either way a dump will be produced.
- S0C7 ABEND - All programs which were converted from OS/VS COBOL must specify the NUMPROC(MIG) compiler option. Using this option will make the COBOL/MVS compiler generate code very similar to the old OS/VS COBOL code. Also the compiler will use numeric techniques when comparing two data items, as was the case with OS/VS COBOL. Two other options are available for sign processing, NUMPROC(PFD) and NUMPROC(NOPFD). If you use either of these two options (NUMPROC(NOPFD) is the default) with a program converted from OS/VS COBOL, some 0C7 conditions will be bypassed. Both of these options perform logical rather than numeric comparisons and both are performance enhancements, with NUMPROC(PFD) providing the greatest benefit. However it is the users responsibility to ensure valid sign positions in the data fields. Most OS/VS COBOL programs do not contain the proper numeric class test (IF numeric ...) and depend on the S0C7 dump to catch invalid data. This is why all programs converted from OS/VS COBOL must be compiled with the NUMPROC(MIG) option to obtain the same results with COBOL/MVS.
- //SYSOUT Usage by LE - LE sends error messages and report data to the //SYSOUT DD statement. If you code a program that sends data to this same DD statement, your data and any errors detected by LE will be in the same file.

  There are 2 ways to separate the LE information from your program data:

  - Change the Select statement in the COBOL program to point to a different DD statement and add a new DD card to the JCL.
  - LE provides users the option of changing the name changing the name of the DD statement where errors and report information will be sent. This is accomplished with the LE MSGFILE option. An example for the PROC CBL3CALL follows.

  ```
  //iiixxx  EXEC  CBL3CALL......,PARM.GO='/MSGFILE(DDMES)'
  //GO.DDMES DD SYSOUT=A
  ```

  A couple of things to look out for - The "/" before the MSGFILE option indicates that all parameters after the / are for LE and not your program. You may use any DD name you want within the (), just make sure you also add the corresponding DD statement.
- In an OCCURS/DEPENDING clause, if a minimum number of occurrences is not specified, COBOL, assumes the minimum is 1. For example, consider the following statement:

  ```
  05 VARLEN-FIELD OCCURS 5 TIMES DEPENDING ON OTHER-FIELD...
  ```

This would be the same as coding:

```
05 VARLEN-FIELD OCCURS 1 TO 5 TIMES DEPENDING ON
   OTHER-FIELD ...
```

- DEBUG statements - If you decide to use the debug statements in your programs, simply placing the code in the program is not enough. You must also use the run-time DEBUG parameter. This is a LE parameter and not a program parameter. It is specified on the program execution JCL card as follows:

```
//   EXEC PGM=COBTEST,PARM='/DEBUG'
```

  Anything before the "/" are program parameters and everything after are LE parameters.

- FILE STATUS 39 ERRORS - The ANSI 85 standards are very strict in this area. The Record Contains clause and the 01 Level Record must be the same as the LRECL in the DCB. Recording mode must also agree between the FD and the actual file. Be aware the 'ADV' compiler option will add one byte to the output file. Also if you read data from a TSO terminal you will need to put the LRECL on the ALLOCATE statement.
- Execution of COBOL/MVS programs interactively - When your program begins execution, a number of runtime modules may need to be accessed from a library called SYS1.SCEERUN. If you try to run your program interactively you will need to allocate this library. If you execute your program in the batch environment using the standard NIH procedures, you should not have a problem. The SYS1.SCEERUN library is referenced by the batch procedures.
- The SORTCNTL DD statement allows the user to modify the order and fields of an invoked sort dynamically at execution time. It can also be used to change or add SORT/MERGE control statements to programs that invoke the SORT/MERGE facility without having to recompile.
- Whenever possible, achieve maximum blocking flexibility by obtaining DCB information from the JCL or volume label rather than coding it directly in the program. The following example shows how a FD designed to define a data set with a blocking factor of 77 is modified to define a data set of any blocking factor.

  specific blocksize:

```
FD CARD-IN
    BLOCK CONTAINS 77 RECORDS
    RECORD CONTAINS 80 CHARACTERS
```

  any blocksize:

```
FD CARD-IN
    BLOCK CONTAINS 0_/ RECORDS
```

7.4 Data Formats for Inter-Language Communication

The following figures show the ways data can be stored. The source language definitions for each data type are given under the COBOL, FORTRAN, and PL/I headings. For more specific information on data formats, consult the appropriate language manuals and the IBM manual *ESA/390 Principles of Operation*, SA22-7201.

The "MACHINE DATA FORMAT" column in the figures below shows a bit breakdown of the data type as stored internally. Bit positions are written vertically under the machine data format symbols they refer to.

CHARACTER

COBOL	FORTRAN	PL/I	TYPE
PIC X(n) DISPLAY 1<=n<=32767	CHARACTER*n 1<=n<=3267	CHAR(n) 1<=n<=32767	Length = n bytes

MACHINE DATA FORMAT				EXAMPLE	
Char 1	Char 2	…	Char n	Value	Internal hex representation
0 0 - 0 7	0 1 - 8 5			ABCD	C1C2C3C4

Figure 2. Character Formats for Inter-Language Communication

FIXED POINT

The fixed point two-word data type, which is available only in COBOL, is simulated through software and requires all data items to be aligned on a word boundary.

The "Range" given in the table indicates the minimum and maximum values numbers can have in all uses of the language. Idiosyncrasies in languages reduce the full range of numbers in some cases even though they are represented the same internally.

Assumed decimal points in COBOL and PL/I are not shown in the table. They are stored in the same way as other numbers; instructions generated by the compilers keep track of the position of the assumed decimal point.

COBOL	FORTRAN	PL/I	TYPE
PIC S9(1-4) COMP (or COMP-4) Range: -9999 to 9999	INTEGER*2 Range: -32768 to 32767	FIXED BIN (1-15,0) Range: -32768 to 32767	Halfword Length = 2 bytes.
PIC S9(5-9) COMP (or COMP-4) Range: -(9)9s to +(9)9s	INTEGER*4 Range: -2147483648 to 2147483647	FIXED BIN 16-31,0) Range: -2147483648 to 2147483647	Fullword Length =4 bytes.
PIC S9(10-18) COMP (or COMP-4) Range: -(18)9s to +(18)9s	-----	-----	Two-word Length = 8 bytes.

Figure 3. Fixed Point Formats for Inter-Language Communication

MACHINE DATA FORMAT		EXAMPLES	
S I 0 0 - 1 0 1 5	Halfword	Value --------- +1234 -1234	Internal hex representation ---------- 04D2 FB2E
S I 0 0 - 3 0 1 1	Fullword	+1234 --------- -1234	000004D2 ---------- FFFFFB2E
S I 0 0 - 6 0 1 3	Two-word	+1234 --------- -1234	0…04D2 ---------- F...FB2E

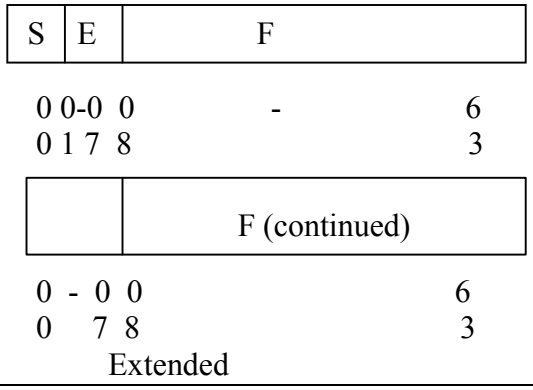
"S" is a binary sign bit: 0 is positive; 1 is negative.
"I" is a 15, 31, or 63 bit integer.
Figure 4 (Continued)

FLOATING POINT
Magnitude is the range of a number expressed in powers of ten.

Although the numbers are represented the same internally, peculiarities in languages reduce the precision of numbers in some cases. The degree of precision given in the table is good in all cases. Fractional precisions occur because of the difference between the decimal representation and the machine's internal storage of numbers.

COBOL	FORTRAN	PL/I	TYPE
COMP-1 Magnitude: 10**-78 to 10**75 Precision: 7.2 digits	REAL*4 Magnitude: 10**-78 to 10**75 Precision: 7.2 digits	FLOAT DEC(1-6) Magnitude: 10**-78 to 10**75 Precision: 6 digits	Short Length = 4 bytes
COMP-2 Magnitude: 10**-78 to 10**75 Precision: 16 digits	REAL*8 Magnitude: 10**-78 to 10**75 Precision: 16.8 digits	FLOAT DEC(7-16) Magnitude: 10**-78 to 10**75 Precision: 16 digits	Long Length = 8 bytes
------	REAL*16 Magnitude: 10**-78 to 10**75 Precision: 35 digits	FLOAT DEC(17-33) Magnitude: 10**-78 to 10**75 Precision: 33 digits	Extended Length = 16 bytes

Figure 4. Floating Point Formats for Inter-Language Communication

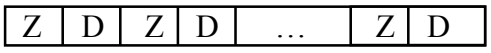
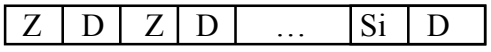
MACHINE DATA FORMAT	EXAMPLES	
	Value ---------- +1234 ---------- -1234	Internal hex representation ---------- 434D2000 ---------- C34D2000
	+1234 ---------- -1234	434D20…0 ---------- C34D20…0
	+1234 ---------- -1234	434D20…0 ---------- C34D20…0

"S" is a binary sign bit: 0 is positive; 1 is negative.
"E" is a seven bit exponent with a value between hex 16** -64 and 16** +63.
"F" is a fraction, which may be 24, 56, or 112 bits long.
Figure 4 (Continued)

ZONED DECIMAL

The "Range" given in the table indicates the minimum and maximum values numbers can have in all uses of the language. Idiosyncrasies in languages reduce the full range of numbers in some cases even though they are represented the same internally.

COBOL	FORTRAN	PL/I	TYPE
PIC 9(n) DISPLAY 1<=n<=18 Range: 0 to (18)9s	------	PIC '(n)9' 1<=n<=15 Range: 0 to (15)9s	Unsigned Length = n bytes.
PIC S9(n) DISPLAY 1<=n<=18 Range: -(18)9s to +(18)9s	------	PIC '(n-1)9T' 1<=n<=15 Range: -(15)9s to +(15)9s	Signed Length = n bytes.

MACHINE DATA FORMAT	EXAMPLES	
 0-0 0-0 0-1 1-1 0 3 4 7 8 1 2 5 Unsigned	Value ---------- 1234	Internal hex representation ---------- F1F2F3F4
 0-0 0-0 0-1 1-1 0 3 4 7 8 1 2 5 Signed	+1234 ---------- -1234	F1F2F3C4 ---------- F1F2F3D4

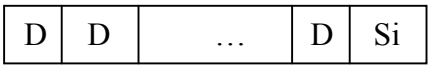
"Z" is a 4 bit zone code with a value of hex F.
"D" is a 4 bit binary decimal number with a value between hex 0 and 9.
"Si" is a 4 bit sign code: A, C, E, and F are positive; B and D are negative.

Figure 5. Zoned Decimal Formats for Inter-Language Communication

PACKED DECIMAL

The "Range" given in the table indicates the minimum and maximum values numbers can have in all uses of the language. Idiosyncrasies in languages reduce the full range of numbers in some cases even though they are represented the same internally.

COBOL	FORTRAN	PL/I	TYPE
COMP-3 PIC 9(n) 1<=n<=18 Range: -(18)9s to +(18)9s	------	FIXED DEC(n) 1<=n<=15 Range: -(15)9s to +(15)9s	Length in bytes = (n+1)/2 rounded up.

MACHINE DATA FORMAT	EXAMPLES	
 0-0 0-0 0 3 4 7	Value ---------- -1234 ---------- -1234	Internal hex representation ---------- 01234C ---------- 01234D

"D" is a 4 bit binary decimal number with a value hex 0 through 9.
"Si" is a 4 bit sign code: A, C, E, and F are positive; B and D are negative.

Figure 6. Packed Decimal Formats for Inter-Language Communication

INDEX

Using COBOL at the NIH Computer Center

Document Evaluation

Is the Manual:

	YES	NO
Clear?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Accurate?	☐	☐
Suitable for the beginner?	☐	☐
Suitable for the advanced user?	☐	☐

Comments:

_____
_____
_____
_____
_____
_____
_____

Please give page references where appropriate. If you wish a reply, include your name and mailing address.

Send to: Application Services Branch
Division of Computer System Services, CIT
National Institutes of Health
Building 12A, Room 4011
Bethesda, MD 20892-5607

FAX to: (301) 496-6905

ICD or Agency:
Date Submitted:
Name (Optional):
E-Mail Address:

manual revised: 9/98